Cost-effective viral marketing in the Latency Aware Independent Cascade model

Robert Gwadera[1] and Grigorios Loukides[2]

[1] Cardiff University GwaderaR@cardiff.ac.uk
[2] King's College London grigorios.loukides@kcl.ac.uk

Abstract. A time-constrained viral marketing campaign allows a business to promote a product or event to social network users within a certain time duration. To perform a time-constrained campaign, existing works select the duration of the campaign, and then a set of k seeds that maximize the *spread* (expected number of users to which the product or event is promoted) for the selected duration. In practice, however, there are many alternative durations, which determine the monetary cost of the campaign and lead to seeds with substantially different spread. In this work, we aim to select the duration of the campaign and a set of k seeds, so that the campaign has the maximum spread-to-cost ratio (i.e., cost-effectiveness). We formulate this task as an optimization problem, under the LAIC information diffusion model. The problem is challenging to solve efficiently, particularly when there are many alternative durations. Thus, we develop an approximation algorithm that employs dynamic programming to compute the spread of seeds for several possible durations simultaneously. We also introduce a new optimization technique that is able to provide an additional performance speed-up by pruning durations that cannot lead to a solution. Experiments on real and synthetic data show the effectiveness and efficiency of our algorithm.

1 Introduction

Many businesses perform time-constrained viral marketing campaigns over social networks, such as Facebook [2, 5, 6, 11, 12]. In these campaigns, a product or event is promoted to a small set of users, who diffuse information about it, with the aim to *activate* their friends (make them aware of the product or event). The active friends of these users diffuse information, attempting to activate their own friends, and the process proceeds similarly, until the end of the campaign duration (e.g., the end of the sales period of the product, or the time the event is held). Typically, the social network is modeled as a graph whose nodes and edges correspond to users and their connections, respectively, and the initial users correspond to a subset of nodes called *seeds*.

Motivation To perform a time-constrained campaign, it is necessary to determine its duration. The duration of the campaign can be modeled as a time window (interval). In practice, the start time of the window is selected by the business based on market properties, such as season, competitors' actions, and availability of products or resources to hold an event [7]. However, there are multiple choices for the end time of the window, which are determined by characteristics of the product and social network. For example, the end time of a campaign that promotes a film corresponds to a time between a few days and five weeks before the release of the film [14]. Therefore, we will assume

a zero start time and model (refer to) each alternative duration as a window defined in terms of its size.

The cost of the campaign, in terms of monetary expense to a business, is a non-decreasing function of the campaign duration. For instance, a social network provider which implements a campaign on a product, as a service to the business [13], charges a fee that increases with the campaign duration. The reason is that multiple businesses compete for performing campaigns simultaneously on the same social network, and executing a campaign with large duration for a product (e.g., a comedy film) reduces the *spread* (expected number of users to which the product is promoted) [11] of other campaigns on substitute products (e.g., different comedy films). Furthermore, the spread of the campaign is also a non-decreasing function of the campaign duration [11].

Thus, a fundamental question for performing a cost-effective campaign is: "Which window (duration) offers the maximum benefit-to-cost ratio?" [16]. The need to perform cost-effective campaigns has been recognized in the marketing literature [3,7,14]. However, the problem has not been studied before. That is, existing methods [2,6,11,12] assume a fixed window that is selected by the business and aim to select a subset of k nodes, as seeds, to maximize the spread for the selected window.

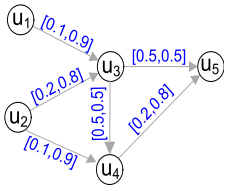
Contributions Our work makes the following contributions:

First, we formulate the *Time-constrained Spread-to-cost Maximization* (TSM) problem, as follows. Given a graph G and a set of candidate windows, each having an associated cost, select: (I) a window, and (II) a subset of k nodes of G as seeds, such that the ratio between the spread of the seeds in the window and the window cost is maximum. In the TSM problem, the spread is computed under the Latency Aware Independent Cascade (LAIC) [11] model. The model takes into account the varying delays (latencies) with which nodes may be activated in practice and generalizes other models [2,8]. Solving TSM allows implementing a cost-effective campaign. However, this is challenging because TSM is NP-hard and cannot be approximated by directly applying the greedy submodular maximization algorithm [15]. This is because the optimization function that computes the maximum spread-to-cost ratio of a seed-set over all windows is not submodular, as we show, whereas the algorithm of [15] requires its optimization function to be submodular. To illustrate TSM, we provide Example 1.

(a)

size (end time)	cost in $
$\|W_1\| = 1$	10
$\|W_2\| = 2$	30

(b)



(c)

nodes	Spread gain W_1	Spread gain W_2
u_1	1.1	2.01
u_2	1.3	**2.877**
u_3	**2**	2.525
u_4	1.2	1.84
u_5	1	1

(d)

seed-set	max. ratio	min. ratio
$W_1.S$	0.33	0.2
$W_2.S$	0.18	0.096

(e)

nodes	Spread gain W_1
$\{u_3, u_1\}$	1
$\{u_3, u_2\}$	**1.05**
$\{u_3, u_4\}$	0.6
$\{u_3, u_5\}$	0.5

Fig. 1: (a) Size and cost of windows W_1 and W_2. (b) Graph and probability vectors of edges (the probabilities reflect how likely a node is activated by its in-neighbor with delay 0 and 1, respectively). (c) Spread gain of the empty seed-set for W_1 and W_2. The gain is caused by adding a node into the seed-set. (d) Bounds for the spread-to-cost ratio of W_1 and W_2, after k iterations. (e) Spread gain of the seed-set $\{u_3\}$ for W_1. The gain is caused by adding a node into the seed-set.

Example 1. A business plans a campaign for a new product, which starts on the day of product launch and can last one or two weeks. This is modeled with the windows W_1

and W_2, whose sizes are shown in Fig. 1a. A social network provider implements the campaign on the graph of Fig. 1b, as a service to the business. Under the LAIC model, each edge (u', u) in Fig. 1b is associated with a vector of probabilities that u is activated by u' with delay 0 and 1, respectively. The social network provider also determines the window costs as shown in Fig. 1a. The business wants to perform the campaign with the largest spread-to-cost ratio and can give away a product to two users, as an incentive to start diffusing information. Thus, the social network provider needs to solve TSM with $k = 2$.

Second, we propose a dynamic programming equation to compute the probability that a node u has been activated in $[0, W.t]$, where $W.t$ is the end time of a window W. The probability is denoted with $P_{[0,W.t]}(u)$ and computed as:

$$P_{[0,W.t]}(u) = 1 - [1 - P_{W.t}(u)] \cdot [1 - P_{[0,W.t-1]}(u)], \quad (1)$$

where $P_{W.t}(u)$ is the probability that u becomes active at $W.t$ and $P_{[0,W.t-1]}(u)$ is the probability that u has been activated before (at any previous time point). In addition, we sum $P_{[0,W.t]}(u)$ over each node u, to compute the spread of a seed-set in W. The spread is computed by a subroutine of our algorithm for TSM. The subroutine is called *DPSC* and computes the exact value of spread, unlike existing algorithms [11, 12].

Third, we propose *MASP*, an approximation algorithm for the TSM problem. The algorithm starts by associating an empty seed-set with each window. Then, it performs k iterations, where k is the input number of seeds. In each iteration j, *MASP*: (I) Computes the *spread gain* of each window's seed-set, for each *available* node (i.e., node that is not contained in the seed-set). The spread gain is the difference in spread, before and after the addition of a node into the seed-set of the window. (II) Adds into each seed-set the node that maximizes the spread gain of the seed-set. (III) Prunes windows that cannot lead to a solution. After that, the algorithm returns the window with the largest spread-to-cost ratio, among all windows, and its associated seed-set. To improve efficiency, *MASP* creates clusters, each containing all windows with the same seed-set, and applies the *Multiple-window spread gain computation* technique to each cluster. In addition, it uses the *Pruning* technique. These techniques are summarized as follows:

Multiple-window spread gain computation. It efficiently computes the spread gain of the seed-set in each window of the cluster, for each available node. To compute the spread gain for an available node, *DPSC* is applied to the cluster and computes the spread of the seed-set in the largest window, W, of the cluster, after adding the node. Since all windows in the cluster have the same seed-set, the spread in every subwindow W' of W, with end time $W'.t$, is also obtained, by summing the probability $P_{[0,W'.t]}(u)$ of each node u, which is computed during the recursion of Eq. 1. Then, the spread gain is calculated for each window as the difference between the spread obtained by *DPSC* and the spread before adding the available node.

Example 2. *MASP* is applied in Example 1 with $k = 2$. Initially, the windows W_1 and W_2 are associated with the empty seed-set, and a single cluster $\{W_1, W_2\}$ is created. Then, the spread gain of the empty seed-set for each available node, u_1 to u_5, is computed. For instance, the spread gain for u_1 is computed as follows. First, *DPSC* is applied to the cluster $\{W_1, W_2\}$ and computes the spread of the seed-set $\{u_1\}$ in the largest window, W_2, of the cluster as $P_{[0,2]}(u_1) + \ldots + P_{[0,2]}(u_5)$. Each of these probabilities is computed recursively using Eq. 1. Thus, the spread in W_1 is also obtained as

$P_{[0,1]}(u_1) + \ldots + P_{[0,1]}(u_5)$. Next, the spread gain in W_1 and in W_2 is calculated as the difference between the spread computed by *DPSC* and the spread before adding u_1.

Pruning. In iteration j, it computes, for each window, the maximum and minimum spread-to-cost ratio that the window can have after k iterations. The maximum ratio is computed for spread equal to the sum of the spread of the seed-set in the window and the spread gain for each of the top $k - j$ (i.e., remaining) available nodes, in terms of spread gain. The minimum ratio is computed for spread equal to the sum of the spread of the seed-set in the window. This corresponds to the worst case, in which the spread gain for each node is zero. Then, each window whose maximum ratio is smaller than the largest minimum ratio of all windows in the current iteration is removed.

Example 3. (continuing from Example 2) *MASP* adds u_3 into the seed-set of W_1, since u_3 maximizes the spread gain in W_1 (see Fig. 1c). The spread of the seed-set $\{u_3\}$ is 2. Thus, the maximum ratio for W_1 is $\frac{2+1.3}{10} = 0.33$. This is because the spread gain caused by u_2, the top available node in terms of spread gain, is 1.3 (see Fig. 1c), and the cost of W_1 is 10. The minimum ratio for W_1 is $\frac{2}{10} = 0.2$. The maximum and minimum ratio for W_2 is computed similarly and is equal to 0.18 and 0.096, respectively. Since the maximum ratio for W_2 is smaller than the largest minimum ratio, W_2 is pruned.

MASP produces a solution whose spread-to-cost ratio is at least $1 - \frac{1}{\mathsf{e}} \approx 63\%$ of that of the optimal solution. This is because it applies the greedy submodular maximization algorithm [15] to the seed-set of each window, using a submodular optimization function that computes the spread in the window. As we show experimentally, our algorithm is both effective and efficient, unlike baselines that are constructed based on the existing methods for maximizing the spread of a given window in the LAIC model [11, 12]. For example, it was at least one order of magnitude faster than a baseline which applies an existing approximation algorithm [11] for finding the subset of k nodes with the maximum spread to each window, and then selects the solution with the largest ratio.

2 Related Work

In [11, 12], the problem of selecting k seeds that maximize the spread in a fixed window was studied under the LAIC model, and the following methods were proposed: *MC*, *ISP*, and *MISP*. These methods select a subset of k nodes as seeds, by iteratively selecting the available node that causes the maximum gain to a spread estimate. *MC* estimates the spread by performing many Monte Carlo simulations of the diffusion process. *ISP* estimates the spread assuming that a node can be activated only by a path which does not share edges with other paths and has probability at least θ to activate the node. *MISP* is a variation of *ISP* that approximates the spread gain, caused by a node, based on the spread of the node and the probability that the out-neighbors of the node are already activated. These methods are not alternatives to the *MASP* algorithm we propose, because they assume a fixed window. On the contrary, there are multiple possible windows in our TSM problem, and the challenge is to compute the spread of seeds over all windows efficiently.

In [4, 10, 13], the problem of seed selection when there are costs associated with nodes was studied. Specifically, in [10], each node has a given cost, while in [4, 13] all nodes have the same cost. Unlike these works, we consider a time-constrained campaign where each window has an associated cost.

3 Background

Preliminaries Let $G(V, E)$ be a directed graph, where V is a set of nodes and E is a set of edges. The set of in-neighbors of a node u is denoted with $n^-(u)$ and has size $|n^-(u)|$, which is referred to as the *in-degree* of u. The set of out-neighbors of u is denoted with $n^+(u)$ and has size $|n^+(u)|$, which is referred to as the *out-degree* of u.

A path $q = [u_1, u_2, \ldots, u_m]$ is an ordered set of nodes, which has *length* $|q| = m - 1$. A path q in which each node is unique (i.e., a path with no cycle) is a *simple* path. A path that starts and ends at the same node is a *cycle* path. We assume simple paths, unless stated otherwise.

Each window W has the following attributes: (I) seed-set $W.S$, (II) end time $W.t$, (III) spread $W.\sigma$, and (IV) spread gain $W.g()$. The spread $W.\sigma$ is defined as $\sigma(W.S, W.t)$, where $\sigma()$ computes the expected number of nodes that are active at time $W.t$, when the seed-set is $W.S$, under the LAIC model [11]. The spread gain $W.g()$ is defined, for a given node u, as $W.g(u) = \sigma(W.S \cup \{u\}, W.t) - \sigma(W.S, W.t)$.

Let U be a universe of elements and 2^U its power set. A set function $f : 2^U \rightarrow \mathbb{R}$ is *non-decreasing*, if $f(X) \leq f(Y)$ for all subsets $X \subseteq Y \subseteq U$, *monotone*, if $f(X) \leq f(X \cup u)$ for each $u \notin X$, and *submodular*, if and only if it satisfies the *diminishing returns* property $f(X \cup \{u\}) - f(X) \geq f(Y \cup \{u\}) - f(Y)$, for all $X \subseteq Y \subseteq U$ and any $u \in U \setminus Y$ [9].

LAIC model In the LAIC model [11], each node is active or inactive. A subset $S \subseteq V$ of nodes, referred to as *seeds*, are active at the initial time 0, and all other nodes are inactive. Each edge has a probability vector $m((u', u)) = [m_0((u', u)), \ldots, m_\delta((u', u))]$, where $m_i((u', u))$ is the probability that the *inactive* node u is activated by its active in-neighbor u' with delay $i \in [0, \delta]$. The probability vectors of edges are selected based on the population targeted by the campaign [11]. For example, in [11], each $m_i((u', u))$ was set to $\mathcal{P}_{u'}(i) \cdot p((u', u))$, where $\mathcal{P}_{u'}$ is a Poisson distribution with a random parameter (mean rate) λ in $[1, 20]$ that is associated with the node u' and $p((u', u)) = \frac{1}{|n^-(u)|}$.

The diffusion process in the LAIC model proceeds as follows. Each seed s tries to activate its out-neighbors at the initial time 0 only and, if multiple seeds have the same out-neighbor, they all try to activate it in arbitrary order. The out-neighbor u of a seed s becomes active at time $1 + i$ with probability $m_i((s, u))$, where the delay i takes each value in $[0, \delta]$. Each out-neighbor that becomes active remains active, and it tries to activate its own inactive out-neighbors. The process proceeds similarly and ends when no new node becomes active.

Let S be a seed-set and $[X_{0,u}, \ldots, X_{t,u}]$ be a sequence of binary variables, such that $X_{j,u} = 1$, if the node u of the graph G becomes active at time j, and $X_{j,u} = 0$ otherwise. For brevity, we denote $P(X_{j,u} = 1)$ with $P_j(u)$ and $\sum_{j \in [0,t]} P_j(u)$ with $P_{[0,t]}(u)$. The expected number of active nodes of G at time t is given by $\sigma(S, t) = \sum_{u \in G} P_{[0,t]}(u)$ [11, 12]. This equation is used in the *DPSC* subroutine.

4 Computing the probability $P_{[0,t]}(u)$

We examine the computation of $P_{[0,t]}(u)$, the probability that a node u has been activated in $[0, t]$. $P_{[0,t]}(u)$ cannot be computed directly using Eq. 1 because $(1 - P_t(u))$, the probability that u does not become active at t, is not given. Thus, we show how to compute $(1 - P_t(u))$ by taking into account each in-neighbor of u which may activate u at t with any possible delay.

Clearly, if the node u is a seed, then $P_{[0,t]}(u) = 1$. Otherwise, $P_{[0,t]}(u)$ is given by Eq. 2:

$$P_{[0,t]}(u) = 1 - \left(\prod_{u' \in n^-(u)} \prod_{i \in [0, min(t-1,\delta)]} [1 - P_{t-1-i}(u') \cdot m_i((u',u))] \right) \cdot (1 - P_{[0,t-1]}(u)) \quad (2)$$

Eq. 2 computes $P_{[0,t]}(u)$ as the probability of the complement of the event "u does not become active at t nor before t". The probability that u does not become active at t is given in the large parentheses, and it takes into account each in-neighbor u' of u and each possible delay i. The probability that u is not active (i.e., has not been activated) before t is given by $1 - P_{[0,t-1]}(u)$. The correctness of Eq. 2 follows from Theorem 1.

Theorem 1. *Let u be a node that is not a seed. Eq. 2 computes the probability u is active at time t, under the LAIC model.*

Proof. (*Sketch*) Let $A_0, \ldots, A_{\delta'}, B$ be sets of in-neighbors of u, such that any node in A_i can activate u at t with delay $i \in [0, \delta']$, and any node in B can activate u before t. The maximum delay δ' is equal to $min(t-1, \delta)$, since u will not become active at t if the delay is larger. Let also E_{A_i} (respectively, E_B) be the event "u became active by at least one node in A_i" (respectively, in B). Clearly, $P_{[0,t]}(u) = P(\cup_{i \in [0,\delta']} E_{A_i} \cup E_B) = 1 - P(\overline{\cup_{i \in [0,\delta']} E_{A_i} \cup E_B})$ and, by DeMorgan's laws and the multiplication rule, $P_{[0,t]}(u) = 1 - P(\overline{E_{A_{\delta'}}} \mid \cap_{i \in [0,\delta'-1]} \overline{E_{A_i}} \cap \overline{E_B}) \cdot P(\cap_{i \in [0,\delta'-1]} \overline{E_{A_i}} \cap \overline{E_B}) = 1 - P(\overline{E_{A_{\delta'}}} \mid \cap_{i \in [0,\delta'-1]} \overline{E_{A_i}} \cap \overline{E_B}) \cdot P(\overline{E_{A_{\delta'-1}}} \mid \cap_{i \in [0,\delta'-2]} \overline{E_{A_i}} \cap \overline{E_B}) \cdot \ldots \cdot P(\overline{E_{A_0}} \mid \overline{E_B}) \cdot P(\overline{E_B})$. The proof follows from: (I) $P(\overline{E_{A_i}} \mid \cap_{j \in [0,i-1]} \overline{E_{A_j}} \cap \overline{E_B}) = \prod_{u' \in n^-(u)} (1 - P_{t-1-i}(u') \cdot m_i((u',u)))$, which holds for each delay $i \in [0, \delta']$. This is because $\overline{E_{A_i}}$ occurs when each in-neighbor u' of u is contained in A_i and fails to activate u; (II) $P(\overline{E_B}) = 1 - P_{[0,t-1]}(u)$, which holds by definition. □

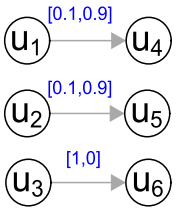
5 The Time-constrained Spread-to-cost Maximization problem

The Time-constrained Spread-to-cost Maximization problem is defined as follows.

Problem (Time-constrained Spread-to-cost Maximization (TSM))**.** *Given the graph $G(V, E)$, the probability vector $m(e)$ of each edge e in E, a set of windows $\mathcal{W} = \{W_1, \ldots, W_n\}$, where each W_i has a nonnegative cost $W_i.c$, and a parameter k, find a window W in $\mathcal{W}$ and a subset $S \subseteq V$ of k nodes, such that the ratio between the spread of S in W and the cost $W.c$ is maximum, over all possible windows of W and their corresponding subsets of k nodes.*

The set of windows $\mathcal{W}$ is determined by the business, based on characteristics of the product and social network [14], while the window costs are determined by the party performing the campaign. TSM is NP-hard, because it generalizes the NP-hard problem in [11], which requires finding a subset of k nodes with maximum spread in a fixed window. The existence of multiple windows makes our problem challenging. For example, we cannot approximate TSM using the greedy algorithm for submodular maximization [15] *with the function $f(S) = max(f_1(S), \ldots, f_n(S))$* (i.e., iteratively add into the seed-set S the node causing the largest gain $f(S \cup \{u\}) - f(S)$), where f_i outputs the spread-to-cost ratio of S in the window W_i. This is because the algorithm of [15] offers approximation guarantees only for submodular functions, whereas f is not submodular (for arbitrary window costs), as shown in Example 4.

Example 4. Consider the graph in Fig. 2a and the windows W_1 and W_2, whose sizes are 1 and 2 and costs are $W_1.c = 1$ and $W_2.c = 1.19$. The spread and the spread-to-cost ratios of different node subsets are shown in Figs. 2b and 2c, respectively. In Fig. 2c, f_1 (resp., f_2) computes the ratio in W_1 (resp., W_2), and the function $f = max(f_1, f_2)$ computes the maximum ratio. The function f is *not* submodular because, for $\{u_3\} \subseteq \{u_2, u_3\}$ and $u_1 \in \{u_1, \ldots, u_6\} \setminus \{u_2, u_3\}$, it holds $f(\{u_3\} \cup \{u_1\}) - f(\{u_3\}) = 3.29 - 2 = 1.29 < f(\{u_2, u_3\} \cup \{u_1\}) - f(\{u_2, u_3\}) = 4.89 - 3.29 = 1.6$.



(a)

node subset	spread	
	W_1	W_2
u_1	1.1	1.91
u_2	1.1	1.91
u_3	2	2
$\{u_1, u_2\}$	2.2	3.82
$\{u_1, u_3\}$	3.1	3.91
$\{u_2, u_3\}$	3.1	3.91
$\{u_1, u_2, u_3\}$	4.2	5.82

(b)

node subset	spread-to-cost ratio		
	f_1	f_2	f
u_1	1.1	1.6	1.6
u_2	1.1	1.6	1.6
u_3	2	1.68	2
$\{u_1, u_2\}$	2.2	3.2	3.2
$\{u_1, u_3\}$	3.1	3.29	3.29
$\{u_2, u_3\}$	3.1	3.29	3.29
$\{u_1, u_2, u_3\}$	4.2	4.89	4.89

(c)

Fig. 2: (a) Graph and probability vectors of edges. (b) The spread of different subsets of nodes of Fig. 2a. (c) The spread-to-cost ratios of different subsets of nodes of Fig. 2a. The ratio in W_1 and W_2 is given by f_1 and f_2, respectively, and the maximum ratio is given by the function f.

6 The *MASP* algorithm

In this section, we present our *MASP* algorithm and its *DPSC* and *Pruning* subroutines.

Algorithm: *MASP* (Multiple-window Addition Spread computation Pruning)
Input: Graph $G(V, E)$, probability vector of each edge of G, set of windows $\mathcal{W}$ and their costs, and parameter k
Output: A window $W \in \mathcal{W}$ and a subset $S \subseteq V$ of k nodes

1 **foreach** *window* W_i *of* $\mathcal{W}$ **do**
2 $W_i.S \leftarrow \varnothing$; $W_i.\sigma \leftarrow 0$
3 Create cluster C comprised of all windows in $\mathcal{W}$
4 Add C into the empty set of clusters $\mathcal{C}$
5 $j \leftarrow 1$ `// iteration counter`
6 **while** $j \leq k$ **do**
7 **foreach** *cluster* C *in* $\mathcal{C}$ **do**
8 $W_C \leftarrow$ the largest window in C
9 **foreach** *node* v *in* $V \setminus W_C.S$ **do**
10 Apply *DPSC* to the cluster C and node v
11 Compute the spread gain $W_i.g(v)$, for each window W_i in C
12 **foreach** *window* W_i *in* $\mathcal{W}$ **do**
13 $u \leftarrow$ the node u in $V \setminus W_i.S$ with the largest spread gain $W_i.g(u)$
14 $W_i.S \leftarrow W_i.S \cup \{u\}$
15 $W_i.\sigma \leftarrow W_i.\sigma + W_i.g(u)$
16 *Pruning*($\mathcal{W}$)
17 $\mathcal{C} \leftarrow$ set of clusters, each containing all windows of $\mathcal{W}$ with the same seed-set
18 $j \leftarrow j + 1$
19 $W \leftarrow$ the window W_i in $\mathcal{W}$ with the maximum $\frac{W_i.\sigma}{W_i.c}$
20 $S \leftarrow$ the seed-set of the window W
21 **return** $\{W, S\}$

MASP initializes, for each window W_i in the given set of windows, its seed-set $W_i.S$ and spread $W_i.\sigma$ (steps 1 to 2). It also initializes a set of clusters $\mathcal{C}$ with a single cluster that contains all windows (steps 3 to 4). Then, it performs k iterations (steps 6 to 18). In each iteration, *MASP*:

I Applies *Multiple-window spread gain computation* to each cluster, to efficiently compute the spread gain $W_i.g(v)$, for each window W_i in the cluster and each available node v (steps 7 to 11). Specifically, the largest window, W_C, in the cluster is found and each node v that is

not contained in the seed-set of W_C is considered. This is without loss of generality, since all windows in the cluster contain the same seed-set. Then, *DPSC* is applied to the cluster and efficiently computes the spread of $W_i.S \cup v$ for every window W_i in the cluster (including W_C). After that, the spread gain $W_i.g(v)$ is computed as the difference between the spread that is obtained from *DPSC*, and the spread $W_i.\sigma$, which was computed before.

II Adds the available node with the largest spread gain into the seed-set $W_i.S$ and updates the spread $W_i.\sigma$, for each window W_i (steps 12 to 15).

III Applies *Pruning* to prune windows that cannot lead to a solution (step 16).

IV Creates a new set of clusters, each containing all windows that are associated with the same seed-set (step 17).

Last, the algorithm finds and returns the window with the largest spread-to-cost ratio, among all windows in $\mathcal{W}$, and its corresponding seed-set (steps 19 to 21).

Theorem 2 explains the approximation guarantee of *MASP*.

Theorem 2. MASP *finds a solution with spread-to-cost ratio at least* $1 - \frac{1}{\mathsf{e}}$ *of that of the optimal solution to the TSM problem, where* e *is the base of the natural logarithm.*

Proof. (*Sketch*) Let σ_i be the maximum spread of a subset of k nodes in a window W_i. *MASP* constructs each seed-set $W_i.S$ using the greedy algorithm for submodular maximization [15] *with the submodular spread function* [11] (i.e., iteratively adds into $W_i.S$ the node u causing the largest spread gain $W_i.g(u)$). This guarantees that, for each W_i, $W_i.\sigma \geq (1 - \frac{1}{\mathsf{e}}) \cdot \sigma_i$ [11]. Thus, for the window with the maximum ratio $max_{i\in[1,n]}W_i.\sigma$, we have $max_{i\in[1,n]}W_i.\sigma \geq (1 - \frac{1}{\mathsf{e}}) \cdot max_{i\in[1,n]}\sigma_i$, which implies $max_{i\in[1,n]}\frac{W_i.\sigma}{W_i.c} \geq (1-\frac{1}{\mathsf{e}}) \cdot max_{i\in[1,n]}\frac{\sigma_i}{W_i.c}$. The proof follows from observing that the spread-to-cost ratio of the solution of *MASP* is $max_{i\in[1,n]}\frac{W_i.\sigma}{W_i.c}$ and that of the optimal solution to TSM is $max_{i\in[1,n]}\frac{\sigma_i}{W_i.c}$. □

MASP needs $O(k \cdot |\mathcal{W}| \cdot |V|^3 \cdot |W_n|)$ time, where $|W_n|$ is the size of the largest window in $\mathcal{W}$, in the worst case in which the graph is complete, all sets contain different seeds in each iteration, and no window is pruned.

Algorithm: DPSC (Dynamic Programming Spread Computation)
Input: Graph $G(V, E)$, probability vector of each edge of G, cluster of windows C, node v
Output: Spread of the seed-set $W_i.S \cup v$, for each window W_i in the cluster C

1 $W_C \leftarrow$ the largest window in C
2 $T \leftarrow$ 2D array with $|V|$ rows and $|W_C|$ columns, with each element equal to zero
3 $\tilde{S} \leftarrow W_C.S \cup v$ `// temporary seed-set`
4 **foreach** *node* s *in the seed-set* $\tilde{S}$ **do**
5 $T[s][0] \leftarrow 1$
6 $t \leftarrow 1$
7 $R \leftarrow reachable(t)$
8 **while** $R \neq \varnothing$ *and time* t *in* W_C **do**
9 **foreach** *node* u *in* R **do**
10 $T[u][t] \leftarrow$ the probability $P_{[0,t]}(u)$
11 **foreach** *node* $u \notin \tilde{S}$ *and* $u \notin R$ *and* u *may have been activated before* t **do**
12 $T[u][t] \leftarrow T[u][t-1]$
13 $t \leftarrow t + 1$
14 $R \leftarrow reachable(t)$
15 **foreach** *window* W_i *in* C **do**
16 $W_i.\tilde{\sigma} \leftarrow \sum_{u\in V} T[u][|W_i|]$ `// spread of the seed-set` $W_i.S \cup v$
17 **return** $\{W_1.\tilde{\sigma}, \ldots, W_C.\tilde{\sigma}\}$

DPSC Given a cluster C and a node v, *DPSC* constructs a temporary seed-set by adding v into the seed-set of the largest window in C (steps 1 to 3), fills a dynamic programming array T, whose element $T[u][t]$ stores the probability $P_{[0,t]}(u)$ for a node

u at time t (steps 4 to 14), and computes $W_i.\tilde{\sigma}$, the spread of the seed-set $W_i.S \cup v$, for each window W_i in C (steps 15 to 16). To improve efficiency, $P_{[0,t]}(u)$ is computed only for the set of nodes that may become active at t, which is found by a function $reachable(t)$. For all other nodes that are not seeds and may have been activated before, $P_{[0,t]}(u)$ is set to $P_{[0,t-1]}(u)$ (steps 11-12). In addition, the probability $P_{t-1-i}(u')$ in Eq. 2 is computed based on the dynamic programming array as $T[u'][t-1-i] - T[u'][t-2-i]$.

The function $reachable(t)$ finds all nodes that are reachable from the seeds through *simple* paths of length $t-i$, for each delay $i \in [0, min(t-1, \delta)]$, using a concurrent breadth-first-search (bfs). The bfs discovers only nodes that may become active at t, which is necessary to accurately compute the probability $P_{[0,t]}(u)$, for each discovered node u. Cycle paths are discarded, because the node u_1 in a cycle path $[u_1, \ldots, u_{m-1}, u_1]$ cannot be activated by the edge (u_{m-1}, u_1).

DPSC needs $O(|V|^2 \cdot |W_C|)$ time, where $|V|$ is the number of nodes of the graph and $|W_C|$ the size of the largest window in C, in the worst case when the graph is complete. In practice, social network graphs are sparse, and *DPSC* scales much better.

Pruning This subroutine prunes windows that cannot lead to a solution of *MASP*. When applied in an iteration j, *Pruning* computes, for each window, the maximum spread-to-cost ratio that the window can have after all remaining $k-j$ iterations (steps 1 to 3). Then, it removes each window whose maximum ratio is smaller than a lower bound, which is computed as the largest spread-to-cost ratio of all windows (steps 4 to 5). The lower bound corresponds to the minimum spread-to-cost ratio of a solution. That is, we assume the worst case, in which every available node in a subsequent iteration is certainly active (i.e., each such node u has spread gain $W_i.g(u) = 0$).

Function: *Pruning*
Input: Set of windows $\mathcal{W}$
1 **foreach** *window* W_i *in* $\mathcal{W}$ **do**
2 $\quad L \leftarrow \text{argmax}_{\{u_1,\ldots,u_{k-j}\} \subseteq V \setminus W_i.S} (\sum_{u \in \{u_1,\ldots,u_{k-j}\}} W_i.g(u))$
3 $\quad W_i.r \leftarrow \frac{W_i.\sigma + \sum_{u \in L} W_i.g(u)}{W_i.c}$ `// max. ratio of` W_i
4 $lbound \leftarrow$ largest ratio $\frac{W_i.\sigma}{W_i.c}$ of each W_i in $\mathcal{W}$
5 Remove from $\mathcal{W}$ each window W_i such that $W_i.r < lbound$

The maximum spread-to-cost ratio, $W_i.r$, of a window W_i is computed based on the following property:

– The spread, $W_i.\sigma$, of W_i cannot increase by more than $\sum_{u \in L} W_i.g(u)$ after any remaining iterations, where L is the set of $k-j$ available nodes with the largest spread gain assigned by $W_i.g()$.

The property holds because, due to the submodularity of spread [11]: (I) no node that is not contained in L can have a larger spread gain than that of a node in L, in any of the remaining $k-j$ iterations of *MASP*, and (II) after the remaining $k-j$ iterations, the spread $W_i.\sigma$ cannot increase by more than the sum of the spread gain of the nodes that are added into $W_i.S$ in the remaining iterations.

7 Experimental evaluation

In this section, we evaluate *MASP* in terms of effectiveness and efficiency and demonstrate the benefit of its optimization techniques. Since no existing algorithms can deal with the TSM problem, we compared *MASP* against three baselines that are based

on the *MC*, *ISP*, and *MISP* methods of [11, 12] (see Section 2). The MC_B baseline applies the *MC* approximation algorithm to each window independently and then selects the solution with the largest spread-to-cost ratio. The ISP_B and $MISP_B$ baselines differ from MC_B in that they estimate the spread using *ISP* and *MISP*, respectively.

All algorithms were implemented in C++ and applied to the datasets in Table 1. All datasets are real and were used in [2, 11, 12], except *AB*, a synthetic dataset generated by the Albert-Barabasi model. *POL* is available at `http://www-personal.umich.edu/~mejn/` and all other real datasets at `http://snap.stanford.edu/data`.

Dataset	Description	# of nodes ($\|V\|$)	# of edges ($\|E\|$)	avg in-degree	max in-degree
WI	Wikipedia adminship vote graph	7115	103689	13.7	452
PH	High Energy Physics citation graph	34546	421578	24.3	846
EPIN	Whom-trusts-whom graph	75879	508837	13.4	3079
POL	Graph of weblogs	1490	19090	11.9	305
AB	Synthetic dataset	10000	45040	9	9997

Table 1: Characteristics of datasets.

Following [11, 12], the probability vector of each edge (u', u) was constructed by setting $p((u', u))$ to $\frac{1}{|n^-(u)|}$ and $\mathcal{P}_{u'}$ to the Poisson distribution with a random parameter (mean rate) λ in $[1, 20]$. In addition, a window set $\mathcal{W}$ of size $|\mathcal{W}|$ was comprised of the windows ending at time $1, \ldots, |\mathcal{W}|$, and δ was set to $|\mathcal{W}| - 1$. The default values for k and $|\mathcal{W}|$ were 25 and 10, respectively. In addition, following [11], we set the number of Monte Carlo simulations in MC_B to 20000, and θ (minimum path probability in *ISP* and *MISP*) to 10^{-5}.

The window costs were assigned by the concave piece-wise linear function in Eq. 3

$$c(W_i) = \begin{cases} |\mathcal{W}| & i = 1 \\ \frac{|\mathcal{W}|}{i} + c(W_{i-1}) & \text{otherwise} \end{cases} \tag{3}$$

Clearly, the cost of a window $c(W_i)$ increases with the end time of the window, but the increase is smaller for larger windows. Concave piece-wise linear functions model "economies of scale" (i.e., the social network provider offers discounts for longer campaigns, which makes it cheaper to extend the length of an already long campaign) [1]. All experiments ran on an Intel Xeon at 2.60GHz with 16GB RAM.

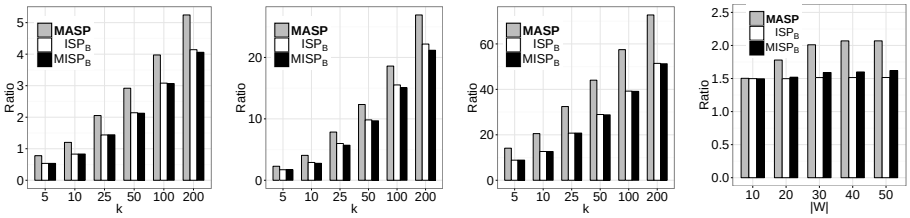


Fig. 3: Spread-to-cost ratio vs. k for (a) *WI*, (b) *PH*, and (c) *EPIN*. Spread-to-cost ratio vs. $|\mathcal{W}|$ for (d) *WI*.

Effectiveness We demonstrate that *MASP* finds solutions with high spread-to-cost ratio, due to its exact spread computation strategy, unlike ISP_B and $MISP_B$. Figs. 3a, 3b, and 3c show the result for varying k. The spread-to-cost ratio for *MASP* was higher than that of both heuristics by 28% on average. Figs. 3d, 4a, and 4b show the spread-to-cost ratio for varying number of windows $|\mathcal{W}|$. The spread-to-cost ratio for *MASP* was higher than that of both heuristics by 32% on average and up to 116%. MC_B found the same solutions with *MASP*, due to the large number of Monte Carlo simulations.

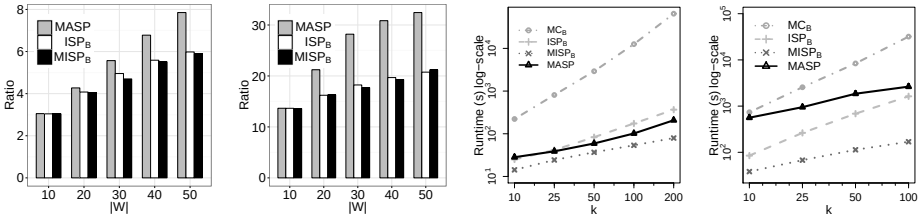


Fig. 4: Spread-to-cost ratio vs. $|\mathcal{W}|$ for (a) *PH*, (b) *EPIN*. Runtime vs. k for (a) *POL*, and (b) *WI*.

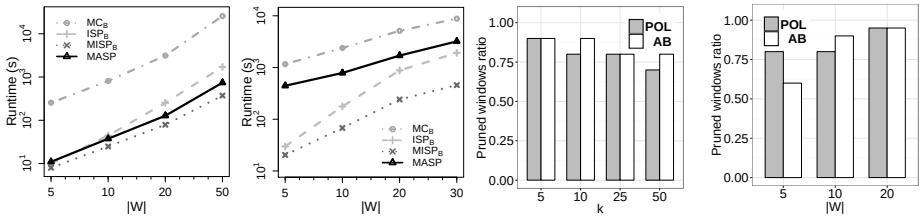


Fig. 5: Runtime vs. $|\mathcal{W}|$ for (a) *POL* and (b) *WI*. Ratio of pruned windows vs. (c) k and (d) $|\mathcal{W}|$, for *POL* and *AB*.

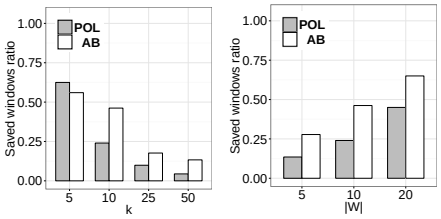


Fig. 6: Ratio of saved windows vs. (a) k and (b) $|\mathcal{W}|$, for the *POL* and *AB* datasets.

Efficiency We demonstrate that *MASP* is significantly faster than MC_B, the only baseline that offers approximation guarantees. Figs. 4c and 4d show the runtime for varying k. *MASP* is *at least* 1, *and on average* 6, *orders of magnitude faster* than MC_B, and it scales much better with respect to k. Figs. 5a and 5b show the runtime for varying number of windows $|\mathcal{W}|$. *MASP* is *at least* 2 *orders of magnitude faster* than MC_B and scales better with respect to $|\mathcal{W}|$. *MASP* is more efficient and scalable than MC_B, due to the pruning and multiple-window spread gain computation, as explained below. However, it is generally less scalable than ISP_B and $MISP_B$.

Thus, the conclusion from the effectiveness and efficiency experiments is that *MASP*: (I) finds the same solutions with MC_B, substantially outperforming ISP_B and $MISP_B$, and (II) is at least one order of magnitude faster than MC_B but less efficient than ISP_B and $MISP_B$.

Benefit of pruning Fig. 5c shows the ratio of pruned windows, for varying k. The ratio is at least 0.7 and 0.8, for the *POL* and *AB* dataset, respectively. Fig. 5d reports the ratio of pruned windows, for varying $|\mathcal{W}|$. The ratio is at least 0.77 and 0.6 for *POL* and *AB* and increases with $|\mathcal{W}|$. This is because more windows have similar ratios, due to the small increase in cost and spread, when $|\mathcal{W}|$ is large.

Benefit of multiple-window spread gain computation We define the ratio of *saved* windows as $\frac{\sum_{i\in[1,k]}\sum_{C\in\mathcal{C}}(|C|-1)}{\sum_{i\in[1,k]}\sum_{C\in\mathcal{C}}|C|}$, where i is an iteration of *MASP* and C is a cluster of windows in the set of clusters $\mathcal{C}$ (see steps 4 and 17 of *MASP*). A *saved* window is not the largest in its cluster and its spread is computed efficiently by *DPSC*. Figs. 6a and 6b show the ratio of *saved* windows for varying k and $|\mathcal{W}|$, respectively. The

ratio decreases with k, because the probability that two windows have the same seed-set decreases with the size of the seed-set. On the other hand, the ratio increases with $|\mathcal{W}|$, because there are more windows that can have the same seed-set and form a cluster.

8 Conclusion

The task of performing a cost-effective, time-constrained campaign requires selecting a window, among given alternatives, and a set of k seeds, such that the ratio between the spread of the seeds in the window and the window cost is maximum. In this work, we formulated this task as an optimization problem and developed an approximation algorithm to solve it. The algorithm employs dynamic programming and pruning to improve efficiency, and it is effective and efficient, as shown experimentally. In the future, we plan to extend the TSM problem when the nodes are also associated with costs.

9 Acknowledgments

The authors would like to thank the reviewers for their constructive comments.

References

1. D. Chen, R. G. Batson, and Y. Dang. *Applied Integer Programming*. 2010.
2. W. Chen, W. Lu, and N. Zhang. Time-critical influence maximization in social networks with time-delayed diffusion process. In *AAAI*, pages 592–598, 2012.
3. D. Cruz and C. Fill. Evaluating viral marketing: isolating the key criteria. *Marketing Intelligence & Planning*, 26(7):743–758, 2008.
4. T. N. Dinh, H. Zhang, D. T. Nguyen, and M. T. Thai. Cost-effective viral marketing for time-critical campaigns in large-scale social networks. *TON*, 22(6):2001–2011, 2014.
5. M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML*, pages 561–568, 2011.
6. M. Gomez-Rodriguez and B. Schölkopf. Influence maximization in continuous time diffusion networks. In *ICML*, pages 1–8, 2012.
7. P. Grifoni, A. D'Andrea, and F. Ferri. An integrated framework for on-line viral marketing campaign planning. *International Business Research*, 6(1):22–30, 2013.
8. D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.
9. A. Krause and D. Golovin. Submodular function maximization. In *Tractability*. 2013.
10. M. van Leeuwen and A. Ukkonen. Same bang, fewer bucks: efficient discovery of the cost-influence skyline. In *SDM*, pages 19–27, 2015.
11. B. Liu, G. Cong, D. Xu, and Y. Zeng. Time constrained influence maximization in social networks. In *ICDM*, pages 439–448, 2012.
12. B. Liu, G. Cong, Y. Zeng, D. Xu, and Y. M. Chee. Influence spreading path and its application to the time constrained social influence maximization problem and beyond. *TKDE*, 26(8):1904–1917, 2014.
13. W. Lu, F. Bonchi, A. Goyal, and L. V. S. Lakshmanan. The bang for the buck: Fair competitive viral marketing from the host perspective. In *KDD*, pages 928–936, 2013.
14. S. Moore. Film Talk: An investigation into the use of viral videos in film marketing. *Journal of Promotional Communications*, 3(3):380–404, 2015.
15. G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.
16. T. L. Saaty. *Decision Making for Leaders*. RWS Publications, 1990.